VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SIGN RECOGNITION FROM VIDEO SEQUENCES

DIGITAL IMAGE PROCESSING
(SWE1010)

SLOT:- A1

Team Members

20MIA1031 - Sanjay.M
20MIA1039 - Sriganth.R
20MIA1117 - Pilaram Manoj

TABLE OF CONTENTS

ABSTRACT

Hand gesture recognition is crucial in human-computer interaction. We present a novel real-time method for hand gesture recognition in this paper. The background subtraction method is used in our framework to extract the hand region from the background. The palm and fingers are then segmented in order to detect and recognise the fingers. Finally, a rule classifier is used to predict hand gesture labels.The translation process, hearing people's integration, and the teaching of hand sign to the hearing community might all benefit from robust automatic identification of hand sign .The rationale behind opting for a vision-based system is that it offers an easier and more natural means of communication between a person and a computer.
The inability to communicate is considered a real illness. People with this condition communicate in a variety of ways, with sign being one of the most popular.Creating sign applications for deaf individuals is critical because they will be able to communicate easily with those who do not understand signs.Creating sign applications for deaf individuals is critical because they will be able to communicate easily with those who do not understand signs.The background subtraction method is used in our framework to extract the hand region from the background. The palm and fingers are then segmented in order to detect and recognise the fingers. Finally, a rule classifier is used to predict hand gesture labels.The translation process, hearing people's integration, and the teaching of hand sign to the hearing community might all benefit from robust automatic identification of hand sign .

INTRODUCTION

As we all know, vision-based hand gesture recognition technology is an important part of human-computer interaction . In recent decades, the keyboard and mouse have played an important role in human-computer interaction. However, new types of HCI methods have been required due to the rapid development of hardware and software. Speech recognition and gesture recognition, in particular, are receiving a lot of attention in the field of HCI.
A gesture is a visual representation of physical behaviour or emotional expression. It includes both body and hand gestures. It is divided into two types static gesture and dynamic gesture. The former denotes a sign by the posture of the body or the gesture of the hand. Movement of the body or the hand conveys messages to the latter. Gesture can be used to communicate between a computer and a human. It differs significantly from traditional hardware-based methods in that it can achieve human-computer interaction through gesture recognition.A gesture is any movement made by a bodily part, such as the hand or the face. Using image processing and computer vision, we can recognise gestures in this case. In addition to helping computers comprehend human gestures, gesture recognition also serves as a translator between computers and people.Like some other oral languages, hand sign is a naturally evolving language. It is used on a regular basis for communication by those who are deaf. Any movement of the body, including the hand is a gesture. Here, we are utilising image processing and computer vision for gesture identification. Gesture recognition makes it possible for computers to comprehend human behaviour and serves as a translator between computers and people. Since there aren't many people who use hand sign in general, finding knowledgeable individuals to record signs is more difficult.Because of these factors, the prevailing consensus is that we are still far from being resilient systems that recognise sign language.If there is any noise in background of live video detection image we are going to use Image processing. we are going to use different types of image processing filters for the images which we detected.

LITERATURE REVIEW

This project will discuss work done in the field of hand gesture recognition. The emphasis here is on soft computing-based primary strategies such as artificial neural networks, fuzzy logics, genetic algorithms, and intelligent approaches. The study also takes into account strategies for hand image construction and preprocessing for segmentation. Many researchers used finger tips to detect hand gestures in appearance, which is primarily based on modeling. Finally, various comparisons of results provided by completely different researchers are provided.The disadvantages of this paper are that we will work in the area of individual finger position bending detection and movements, which has received little attention.
An Analysis of Features for Hand-Gesture Classification:
Human-computer interaction, or HCI, is entirely dependent on physical devices. The goal of this work is to evaluate and analyse methods that allow users to interact with machines using natural language-based hand gestures. We present some hand gesture approaches used in HCI systems, as well as a replacement proposal that uses geometric shape descriptors for hand gesture classification. The analysis of the results shows that this new proposal overcomes some of the limitations of various known HCI methods.The disadvantages of this paper are that the user must wear special gloves in order to measure the hand pose and joint angles. The issue with this technique is that once the user wears a glove, the system becomes invasive, in addition to the fact that special gloves are expensive.

OVERVIEW OF PROJECT

Figure 1 depicts an overview of hand gesture recognition. First, the hand is detected using the background subtraction method, and the resulting binary image is transformed. The fingers and palm are then segmented to aid in finger recognition. Additionally, the fingers are detected and recognised. Finally, a simple rule classifier is used to recognise hand gestures.

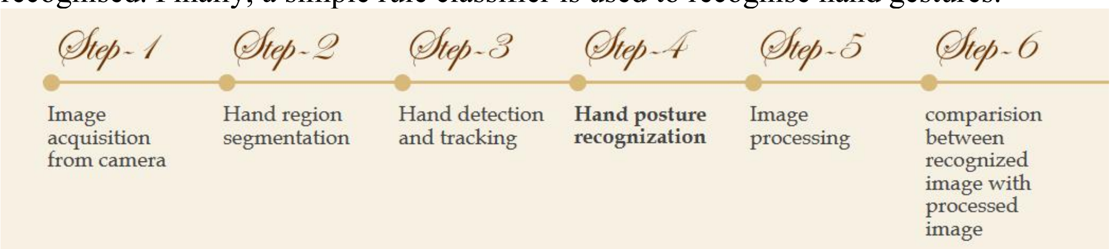


Figure 1

Hand Detection:
It shows the original images used in the work for hand gesture recognition. These images were taken with a standard camera. These hand images were all taken under the same conditions. The backgrounds of these images are identical. As a result, using the background subtraction method, it is simple and effective to detect the hand region from the original image. However, in some cases, other moving objects are included in the result of background subtraction. The skin colour can be used to distinguish the hand region from other moving objects. The HSV model is used to determine skin colour. The HSV (hue, saturation, and value) values of the skin colours respectively. To make the gesture recognition invariant to image scale, the image of the detected hand is resize.

CODING OF HAND RECOGNISATION

Gesture_Controller.py

```python
# Imports
import time
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol

pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Gesture Encodings
class Gest(IntEnum):
    # Binary Encoded
    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31
```

Gesture_Controller.py

```python
    # Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1

# Convert Mediapipe Landmarks to recognizable Gestures
class HandRecog:

    def __init__(self, hand_label):
        self.finger = 0
        self.ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

    def update_hand_result(self, hand_result):
        self.hand_result = hand_result

    def get_signed_dist(self, point):
        sign = -1
        if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x)**2
```

Gesture_Controller.py

```python
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist*sign

    def get_dist(self, point):
        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist

    def get_dz(self,point):
        return abs(self.hand_result.landmark[point[0]].z - self.hand_result.landmark[point[1]].z)

    # Function to find Gesture Encoding using current finger_state.
    # Finger_state: 1 if finger is open, else 0
    def set_finger_state(self):

        if self.hand_result == None:
            return

        points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
        self.finger = 0
        self.finger = self.finger | 0 #thumb
        for idx,point in enumerate(points):

            dist = self.get_signed_dist(point[:2])
            dist2 = self.get_signed_dist(point[1:])

            try:
                ratio = round(dist/dist2,1)
```

Gesture_Controller.py

```python
            except:
                ratio = round(dist/0.01,1)

            self.finger = self.finger << 1
            if ratio > 0.5 :
                self.finger = self.finger | 1


    # Handling Fluctations due to noise
    def get_gesture(self):

        if self.hand_result == None:
            return Gest.PALM

        current_gesture = Gest.PALM
        if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4]) < 0.05:
            if self.hand_label == HLabel.MINOR :
                current_gesture = Gest.PINCH_MINOR
            else:
                current_gesture = Gest.PINCH_MAJOR

        elif Gest.FIRST2 == self.finger :
            point = [[8,12],[5,9]]
            dist1 = self.get_dist(point[0])
            dist2 = self.get_dist(point[1])
            ratio = dist1/dist2
            if ratio > 1.7:
                current_gesture = Gest.V_GEST
            else:
                if self.get_dz([8,12]) < 0.1:
                    current_gesture =  Gest.TWO_FINGER_CLOSED
                else:
                    current_gesture =  Gest.MID

        else:
            current_gesture =  self.finger

        if current_gesture == self.prev_gesture:
            self.frame_count += 1
        else:
            self.frame_count = 0

        self.prev_gesture = current_gesture

        if self.frame_count > 4 :
            self.ori_gesture = current_gesture
        return self.ori_gesture

# Executes commands according to detected gestures
class Controller:

    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = int(sbcontrol.get_brightness(display=0))/100.0
        currentBrightnessLv += Controller.pinchlv/50.0
        if currentBrightnessLv > 1.0:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < 0.0:
            currentBrightnessLv = 0.0
        sbcontrol.fade_brightness(int(100*currentBrightnessLv) , start = sbcontrol.get_brightness(display=0))

    def changesystemvolume(self):
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv/50.0
```

Gesture_Controller.py

```python
        x_old,y_old = pyautogui.position()
        x = int(position[0]*sx)
        y = int(position[1]*sy)
        if Controller.prev_hand is None:
            Controller.prev_hand = x,y
        delta_x = x - Controller.prev_hand[0]
        delta_y = y - Controller.prev_hand[1]

        distsq = delta_x**2 + delta_y**2
        ratio = 1
        Controller.prev_hand = [x,y]

        if distsq <= 25:
            ratio = 0
        elif distsq <= 900:
            ratio = 0.07 * (distsq ** (1/2))
        else:
            ratio = 2.1
        x , y = x_old + delta_x*ratio , y_old + delta_y*ratio
        return (x,y)

    def pinch_control_init(hand_result):
        Controller.pinchstartxcoord = hand_result.landmark[8].x
        Controller.pinchstartycoord = hand_result.landmark[8].y
        Controller.pinchlv = 0
        Controller.prevpinchlv = 0
        Controller.framecount = 0

    # Hold final position for 5 frames to change status
    def pinch_control(hand_result, controlHorizontal, controlVertical):
```

Gesture_Controller.py

```python
    def changesystemvolume(self):
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv/50.0
        if currentVolumeLv > 1.0:
            currentVolumeLv = 1.0
        elif currentVolumeLv < 0.0:
            currentVolumeLv = 0.0
        volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

    def scrollVertical(self):
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)


    def scrollHorizontal(self):
        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')

    # Locate Hand to get Cursor Position
    # Stabilize cursor by Dampening
    def get_position(hand_result):

        point = 9
        position = [hand_result.landmark[point].x ,hand_result.landmark[point].y]
        sx,sy = pyautogui.size()
```

Gesture_Controller.py

```python

        if Controller.framecount == 5:
            Controller.framecount = 0
            Controller.pinchlv = Controller.prevpinchlv

            if Controller.pinchdirectionflag == True:
                controlHorizontal() #x

            elif Controller.pinchdirectionflag == False:
                controlVertical() #y

        lvx =  Controller.getpinchxlv(hand_result)
        lvy =  Controller.getpinchylv(hand_result)

        if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = False
            if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvy
                Controller.framecount = 0

        elif abs(lvx) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = True
            if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvx
                Controller.framecount = 0

```

Gesture_Controller.py

```python
    def handle_controls(gesture, hand_result):
        x,y = None,None
        if gesture != Gest.PALM :
            x,y = Controller.get_position(hand_result)

        # flag reset
        if gesture != Gest.FIST and Controller.grabflag:
            Controller.grabflag = False
            pyautogui.mouseUp(button = "left")

        if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
            Controller.pinchmajorflag = False

        if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
            Controller.pinchminorflag = False

        # implementation
        if gesture == Gest.V_GEST:
            Controller.flag = True
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.FIST:
            if not Controller.grabflag :
                Controller.grabflag = True
                pyautogui.mouseDown(button = "left")
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.MID and Controller.flag:
            pyautogui.click()
            Controller.flag = False
```

Gesture_Controller.py

```python
                pyautogui.mouseDown(button = "left")
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.MID and Controller.flag:
            pyautogui.click()
            Controller.flag = False

        elif gesture == Gest.INDEX and Controller.flag:
            pyautogui.click(button='right')
            Controller.flag = False

        elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
            pyautogui.doubleClick()
            Controller.flag = False

        elif gesture == Gest.PINCH_MINOR:
            if Controller.pinchminorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchminorflag = True
            Controller.pinch_control(hand_result,Controller.scrollHorizontal, Controller.scrollVertical)

        elif gesture == Gest.PINCH_MAJOR:
            if Controller.pinchmajorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchmajorflag = True
            Controller.pinch_control(hand_result,Controller.changesystembrightness, Controller.changesystemvolume)

class GestureController:

    gc_mode = 0
```

Gesture_Controller.py

```python
    def __init__(self):
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT = GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH = GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)

    def classify_hands(results):

        left_, right = None,None
        try:
            handedness_dict = MessageToDict(results.multi_handedness[0])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[0]
            else_:
                left = results.multi_hand_landmarks[0]
        except:
            pass

        try:
            handedness_dict = MessageToDict(results.multi_handedness[1])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[1]
            else_:
                left = results.multi_hand_landmarks[1]
        except:
            pass

        if GestureController.dom_hand == True:
            GestureController.hr_major = right
```

Gesture_Controller.py

```python
        if GestureController.dom_hand == True:
            GestureController.hr_major = right
            GestureController.hr_minor = left
        else_:
            GestureController.hr_major = left
            GestureController.hr_minor = right

    def start(self):

        handmajor = HandRecog(HLabel.MAJOR)
        handminor = HandRecog(HLabel.MINOR)

        with mp_hands.Hands(max_num_hands_=_2,min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
            while GestureController.cap.isOpened() and GestureController.gc_mode:
                success, image = GestureController.cap.read()

                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
                image.flags.writeable = False
                results = hands.process(image)

                image.flags.writeable = True
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)
```

Gesture_Controller.py

```python
                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)
                    handmajor.update_hand_result(GestureController.hr_major)
                    handminor.update_hand_result(GestureController.hr_minor)
                    for i in range(0, 8):
                        # cv2.imwrite('python_img', image)
                        cv2.imwrite("image%04i.jpg" % i, image)
                        cv2.waitKey(5)
                    handmajor.set_finger_state()
                    handminor.set_finger_state()
                    gest_name = handminor.get_gesture()

                    if gest_name == Gest.PINCH_MINOR:
                        Controller.handle_controls(gest_name, handminor.hand_result)
                    else:
                        gest_name = handmajor.get_gesture()
                        Controller.handle_controls(gest_name, handmajor.hand_result)

                    for hand_landmarks in results.multi_hand_landmarks:
                        mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
                else:
                    Controller.prev_hand = None
                cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5) & 0xFF == 13:
                    break
        GestureController.cap.release()
        cv2.destroyAllWindows()
# uncomment to run directly
gc1 = GestureController()
gc1.start()
```

RESULTS OF HAND RECOGNISATION

IMAGE PROCESSING TECHIQUES USED

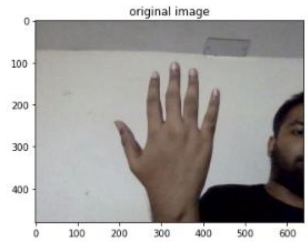
- **Low pass filter:** Smoothing an image involves reducing the disparity between pixel values by averaging nearby pixels. When using a low pass filter, the low frequency information within an image is retained while the high frequency information is reduced.
- **High pass filter:** contrast between adjacent areas with little variation in brightness or darkness is increased, an image is sharpened. A high pass filter tends to retain high frequency information while reducing low frequency information in an image.
- **Slat and pepper:** Salt and pepper noise refers to a wide range of processes that all result in the same fundamental image degradation: only a few pixels are noisy, but they are very noisy. The effect is similar to sprinkling salt and pepper (white and black dots) on the image.
- **Median filter:** The median filter is a type of non-linear digital filter that is commonly used to remove noise from an image or signal. The median filter is very important in image processing because it is well known for preserving edges during noise removal.
- **Laplacian filter** This specifies whether a change in adjacent pixel values is caused by an edge or by continuous progression.The surface Laplacian can reduce spatial noise and improve prediction when used as a spatial filter.
- **Average mean** filtering is a technique for'smoothing' images by reducing the intensity variation between adjacent pixels. The average filter operates by pixel-by-pixel traversal of the image, replacing each value with the average value of neighbouring pixels, including itself.
- **Sobel filter:** The Sobel edge detector employs a pair of 3 3 convolution masks, one for estimating the gradient in the x-direction and the other for estimating the gradient in the y-direction.
- **RGB:** The RGB color model is an additive color model in which the red, green, and blue primary colors of light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.
- **Grayscale**: Grayscale refers to a situation in which each pixel in a digital image solely contains information about the light's intensity. Usually, just the range from deepest black to brightest white is visible in photos.
- negative image is a total inversion of a positive image, in which light. areas appear dark and vice versa

CODING & RESULT OF IMAGE PROCESSING

In [3]:
```
%pylab inline
from __future__ import division
from __future__ import print_function
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [4]:
```
import matplotlib.pyplot as plt
from PIL import Image
import matplotlib.image as mpimg
A = mpimg.imread('G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.3.jpg')
imgplot = plt.imshow(A),plt.title('original image')
plt.show()
```

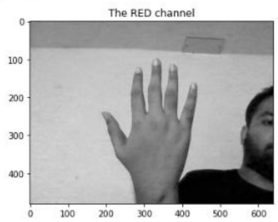


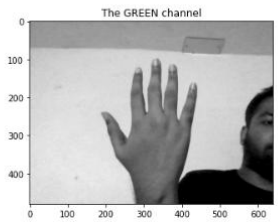
In [5]:
```
print(np.shape(A))
print(type(A))
print(A.dtype)
```

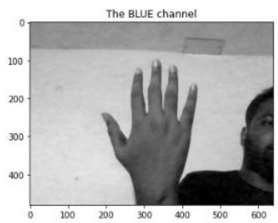
```
(480, 640, 3)
<class 'numpy.ndarray'>
uint8
```

In [6]:
```
A_red = A[:, :, 0]
A_green = A[:, :, 1]
A_blue = A[:, :, 2]

plt.figure()
plt.imshow(A_red, cmap=cm.gray)
plt.title('The RED channel')

plt.figure()
plt.imshow(A_green, cmap=cm.gray)
plt.title('The GREEN channel')
```

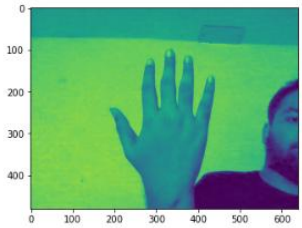
```
plt.figure()
plt.imshow(A_green, cmap=cm.gray)
plt.title('The GREEN channel')

plt.figure()
plt.imshow(A_blue, cmap=cm.gray)
plt.title('The BLUE channel')

plt.show()
```







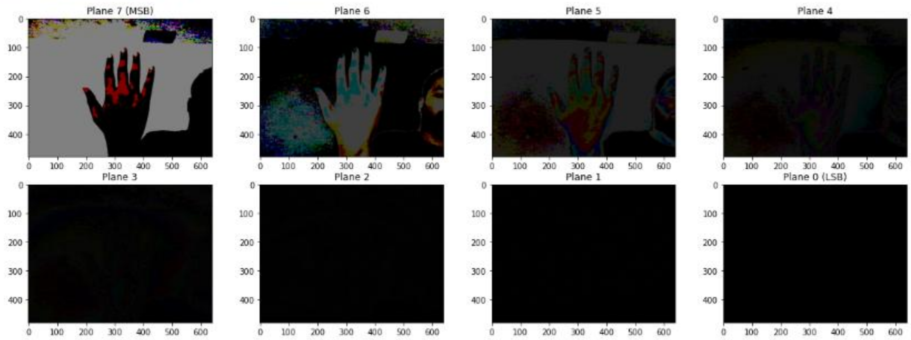
In [7]:
```
plt.figure()
plt.imshow(A_red)
plt.show()
```



In [8]:
```
A_gr = plt.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.3.jpg")
```

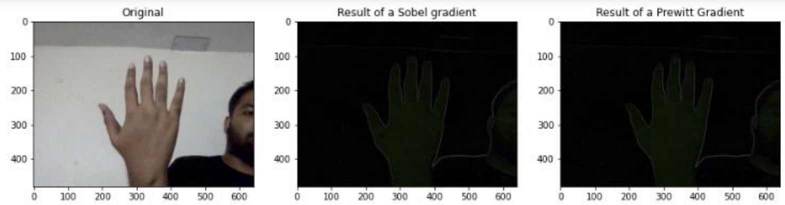
In [9]:
```
plane7 = A_gr & 128*np.ones(shape(A_gr)).astype('uint8')
plane6 = A_gr &  64*np.ones(shape(A_gr)).astype('uint8')
plane5 = A_gr &  32*np.ones(shape(A_gr)).astype('uint8')
plane4 = A_gr &  16*np.ones(shape(A_gr)).astype('uint8')
plane3 = A_gr &   8*np.ones(shape(A_gr)).astype('uint8')
plane2 = A_gr &   4*np.ones(shape(A_gr)).astype('uint8')
plane1 = A_gr &   2*np.ones(shape(A_gr)).astype('uint8')
plane0 = A_gr &   1*np.ones(shape(A_gr)).astype('uint8')

plt.figure(figsize=(20,7))
plt.subplot(2, 4, 1)
plt.imshow(plane7, cmap=cm.gray)
plt.title('Plane 7 (MSB)')
plt.subplot(2, 4, 2)
plt.imshow(plane6, cmap=cm.gray)
plt.title('Plane 6')
plt.subplot(2, 4, 3)
plt.imshow(plane5, cmap=cm.gray)
plt.title('Plane 5')
plt.subplot(2, 4, 4)
plt.imshow(plane4, cmap=cm.gray)
plt.title('Plane 4')
plt.subplot(2, 4, 5)
plt.imshow(plane3, cmap=cm.gray)
plt.title('Plane 3')
plt.subplot(2, 4, 6)
plt.imshow(plane2, cmap=cm.gray)
plt.title('Plane 2')
plt.subplot(2, 4, 7)
plt.imshow(plane1, cmap=cm.gray)
plt.title('Plane 1')
plt.subplot(2, 4, 8)
plt.imshow(plane0, cmap=cm.gray)
plt.title('Plane 0 (LSB)')
```

Out[9]: Text(0.5, 1.0, 'Plane 0 (LSB)')



In [10]:
```
from skimage import filters, data
camera = plt.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.3.jpg")

#apply sobel gradient
sobel_camera = filters.sobel(camera)

#apply prewitt gradient
prewitt_camera = filters.prewitt(camera)

figure(figsize=(15, 5))
subplot(1, 3, 1)
imshow(camera, cmap=cm.gray)
title('Original')
subplot(1, 3, 2)
imshow(sobel_camera, cmap=cm.gray)
title('Result of a Sobel gradient')
subplot(1, 3, 3)
imshow(prewitt_camera, cmap=cm.gray)
title('Result of a Prewitt Gradient')
```

Out[10]: Text(0.5, 1.0, 'Result of a Prewitt Gradient')

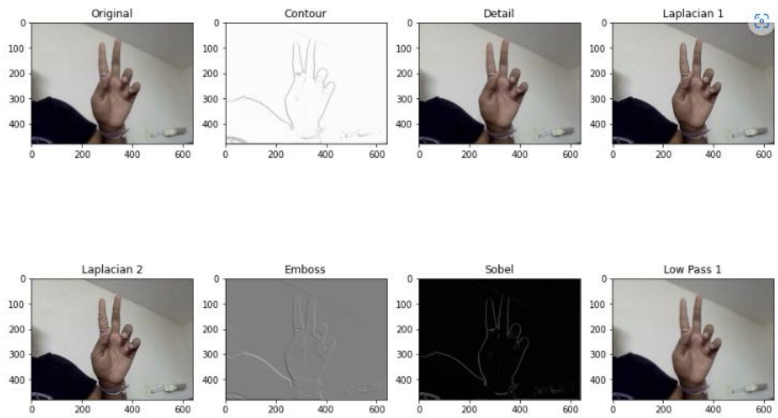


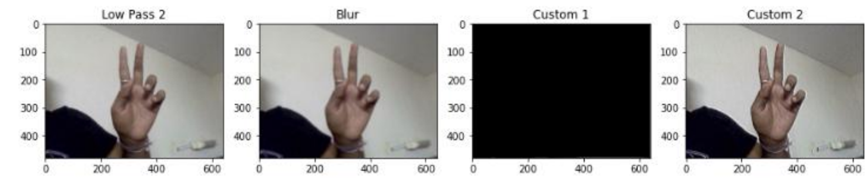
In [11]:
```
from PIL import Image
from PIL import ImageFilter

im0 = Image.open("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.jpg")

figure(figsize=(15,15))
subplot(3,4,1)
plt.imshow(im0)
plt.title('Original')
subplot(3,4,2)
im2 = im0.filter(ImageFilter.CONTOUR)
plt.imshow(im2)
plt.title('Contour')
subplot(3,4,3)
im3 = im0.filter(ImageFilter.DETAIL)
plt.imshow(im3)
plt.title('Detail')
subplot(3,4,4)
im4 = im0.filter(ImageFilter.EDGE_ENHANCE)
plt.imshow(im4)
plt.title('Laplacian 1')
subplot(3,4,5)
im5 = im0.filter(ImageFilter.EDGE_ENHANCE_MORE)
plt.imshow(im5)
plt.title('Laplacian 2')
subplot(3,4,6)
im6 = im0.filter(ImageFilter.EMBOSS)
plt.imshow(im6)
plt.title('Emboss')
subplot(3,4,7)
im7 = im0.filter(ImageFilter.FIND_EDGES)
plt.imshow(im7)
plt.title('Sobel')
subplot(3,4,8)
im8 = im0.filter(ImageFilter.SMOOTH)
plt.imshow(im8)
plt.title('Low Pass 1')
subplot(3,4,9)
im9 = im0.filter(ImageFilter.SMOOTH_MORE)
plt.imshow(im9)
plt.title('Low Pass 2')
subplot(3,4,10)
im10 = im0.filter(ImageFilter.SHARPEN)
```

```
im1 = im0.filter(ImageFilter.BLUR)
plt.imshow(im1)
plt.title('Blur')

#Custom mask

size = (3, 3)
kernel1 = [1, 1, 1, 0, 0, 0, -1, -1, -1]
ker1 = ImageFilter.Kernel(size, kernel1, scale=None, offset=0)
subplot(3,4,11)
im11 = im0.filter(ker1)
plt.imshow(im11)
plt.title('Custom 1')

kernel2 = [1, 0, -1, 1, 0, -1, 0, 0, -1]
ker2 = ImageFilter.Kernel(size, kernel2, scale=None, offset=0)
subplot(3,4,12)
im12 = im0.filter(ker2)
plt.imshow(im12)
plt.title('Custom 2')
```

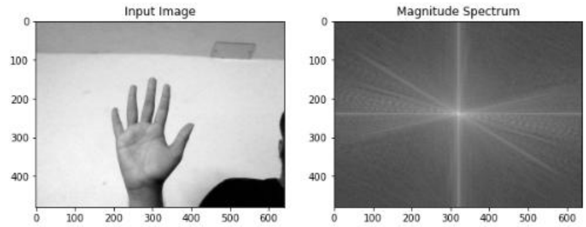
```
<ipython-input-11-c98679439c04>:46: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes cu
rrently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this
warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  subplot(3,4,10)
```

Out[11]: Text(0.5, 1.0, 'Custom 2')

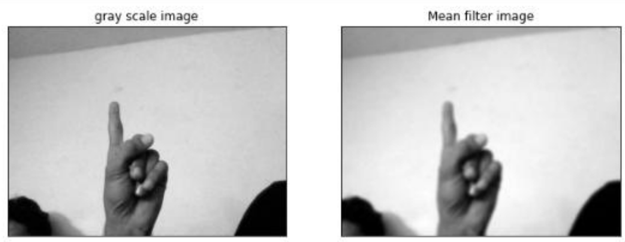


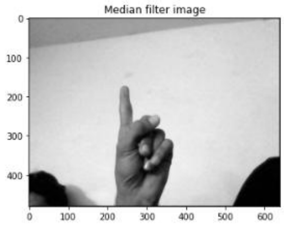


In [12]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.1.jpg",0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.figure(figsize=(10,5))
plt.subplot(121)
plt.imshow(img, cmap = 'gray')
plt.title('Input Image')
plt.subplot(122)
plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum')
plt.show()
```

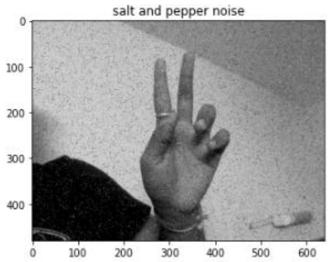


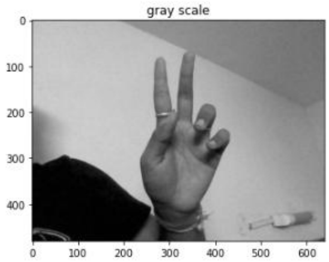
In [13]:
```
#grayscale mean median
image = cv2.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.2.jpg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
image2 = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
figure_size = 9
new_image = cv2.blur(image2,(figure_size, figure_size))
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(image2, cmap='gray'),plt.title('gray scale image')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(new_image, cmap='gray'),plt.title('Mean filter image')
plt.xticks([]), plt.yticks([])
plt.show()
img_noisy1 = cv2.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.2.jpg", 0)
m, n = img_noisy1.shape
img_new1 = np.zeros([m, n])
for i in range(1, m-1):
    for j in range(1, n-1):
        temp = [img_noisy1[i-1, j-1],
               img_noisy1[i-1, j],
               img_noisy1[i-1, j + 1],
               img_noisy1[i, j-1],
               img_noisy1[i, j],
               img_noisy1[i, j + 1],
               img_noisy1[i + 1, j-1],
               img_noisy1[i + 1, j],
               img_noisy1[i + 1, j + 1]]

        temp = sorted(temp)
        img_new1[i, j]= temp[4]

img_new1 = img_new1.astype(np.uint8)
cv2.imwrite('new_median_filtered.png', img_new1)
plt.imshow(img_new1, cmap='gray'),plt.title('Median filter image')
plt.show()
```

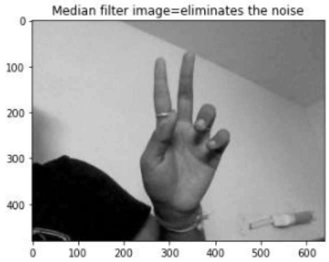




In [14]:
```
#salt and pepper

import random
def add_noise(img):
    row , col = img.shape
    number_of_pixels = random.randint(300, 10000)
    for i in range(number_of_pixels):
        y_coord=random.randint(0, row - 1)
        x_coord=random.randint(0, col - 1)
        img[y_coord][x_coord] = 255
    number_of_pixels = random.randint(3000 , 10000)
    for i in range(number_of_pixels):
        y_coord = random.randint(0, row - 1)
        x_coord=random.randint(0, col - 1)
        img[y_coord][x_coord] = 0
    return img

img = cv2.imread('G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.jpg',
                 cv2.IMREAD_GRAYSCALE)
plt.imshow(add_noise(img), cmap='gray', vmin=0, vmax=255),plt.title('salt and pepper noise')
plt.show()

file = 'G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.jpg'
image = Image.open(file).convert("L")
plt.show()

arr = np.asarray(image)
plt.imshow(arr, cmap='gray', vmin=0, vmax=255),plt.title('gray scale')
plt.show()

img_new1 = img_new1.astype(np.uint8)
cv2.imwrite('new_median_filtered.png', img_new1)
plt.imshow(img_new1, cmap='gray'),plt.title('Median filter image=eliminates the noise')
plt.show()
```

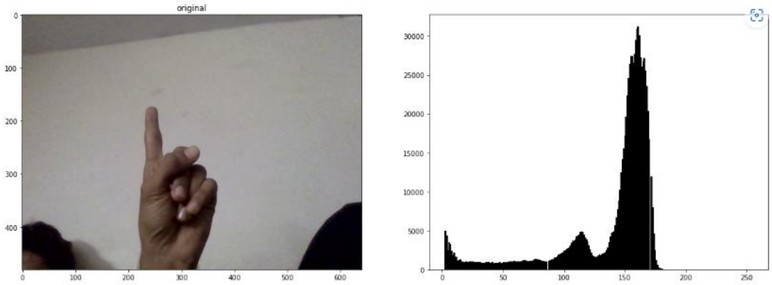


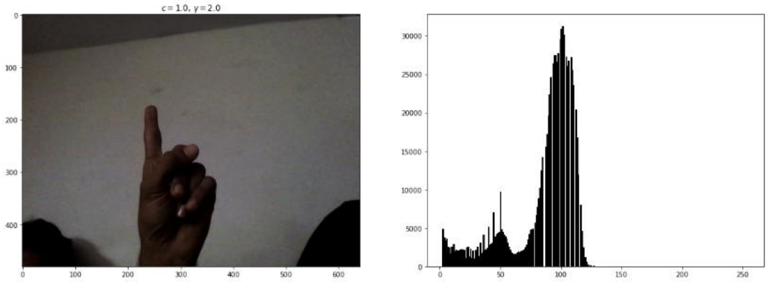


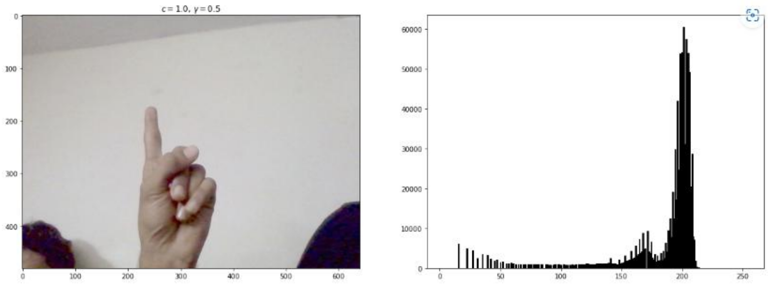


In [15]:
```
xray_orig = plt.imread("G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.2.jpg")
figure(figsize=(20,7))
subplot(1, 2, 1)
plt.imshow(xray_orig, cmap=cm.gray)
plt.title('original')
subplot(1, 2, 2)
plt.hist(xray_orig.flatten(), 256, range=(2, 255), fc='k', ec='k');
```

```
c = 1.0
gamma = 2.0
figure(figsize=(20,7))
subplot(1, 2, 1)
xray_gamma1 = (255*c*(xray_orig / 255)**gamma).astype('uint8')
plt.imshow(xray_gamma1, cmap=cm.gray)
plt.title('$c=1.0$, $\gamma = 2.0$')
subplot(1, 2, 2)
plt.hist(xray_gamma1.flatten(), 256, range=(2, 255), fc='k', ec='k');

c = 1.0
gamma = 0.5
figure(figsize=(20,7))
subplot(1, 2, 1)
xray_gamma2 = (255*c*(xray_orig / 255)**gamma).astype('uint8')
plt.imshow(xray_gamma2, cmap=cm.gray)
plt.title('$c=1.0$, $\gamma = 0.5$')
subplot(1, 2, 2)
plt.hist(xray_gamma2.flatten(), 256, range=(2, 255), fc='k', ec='k');
```







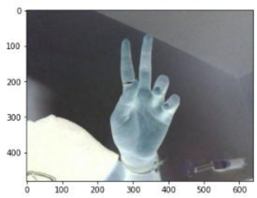
In [16]:
```
girl = plt.imread('G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\dip\\project\\images\\image0006.jpg')
plt.imshow(girl)
maxi = np.amax(girl)
mini = np.amin(girl)
intensity_range = maxi - mini
print('lowest intensity:', mini, ', highest intensity:', maxi, ', spread:', intensity_range)
girl_high = ((girl.astype('float64') - mini) * 255 / intensity_range).astype('uint8')
plt.imshow(girl_high, cmap=cm.gray)
```

```
lowest intensity: 0 , highest intensity: 255 , spread: 255
```

Out[16]: `<matplotlib.image.AxesImage at 0x1b39824a520>`

In [17]:
```
girl_neg = (255*np.ones(shape(girl_high)) - girl_high).astype('uint8')
plt.imshow(girl_neg, cmap=cm.gray)
```

Out[17]: `<matplotlib.image.AxesImage at 0x1b398393d90>`



CONCLISION

In this project, we can able to know that it is very difficult to communicate for the deaf and dumb people. so,by using hand gesture recognition we are going to detect the image from live video from that we can get original image.Image processing and computer vision are used here to identify gestures. Gesture recognition allows computers to understand human behaviour and acts as a translator between computers and humans.We employed image processing here since we don't know the environmental situation that led to the usage of the project, so we applied some image processing based on the situation and image type to communicate the message appropriately by utilising a computer.

References

- https://ieeexplore.ieee.org/document/8663054
- https://www.hindawi.com/journals/tswj/2014/267872/
- https://www.researchgate.net/publication/324394979_Real-Time_Sign_Language_Gesture_Word_Recognition_from_Video_Sequences_Using_CNN_and_RNN
- https://www.researchgate.net/publication/331750214_Dynamic_Sign_Language_Recognition_Based_on_Video_Sequence_with_BLSTM-3D_Residual_Networks